(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
26 July 2001 (26.07.2001)

PCT

(10) International Publication Number
WO 01/54370 A2

(51) International Patent Classification⁷: H04L 29/00

(21) International Application Number: PCT/CA01/00056

(22) International Filing Date: 24 January 2001 (24.01.2001)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
60/177,361 24 January 2000 (24.01.2000) US

(71) Applicant *(for all designated States except US)*: THE UNIVERSITY OF MANITOBA [CA/CA]; 202 Administration Building, Winnipeg, Manitoba R3L 1X3 (CA).

(72) Inventors; and

(75) Inventors/Applicants *(for US only)*: MCLEOD, Robert, D. [CA/CA]; 37 Fifth Avenue, Winniped, Manitoba R2M 0A9 (CA). BLIGHT, David, C. [CA/CA]; 179 Edgewater Drive, Winnipeg, Manitoba R2J 2V4 (CA). KRETSCHMANN, Steven, R. [CA/CA]; 20-117 Bryce Street, Winnipeg, Manitoba R3L 1X3 (CA).

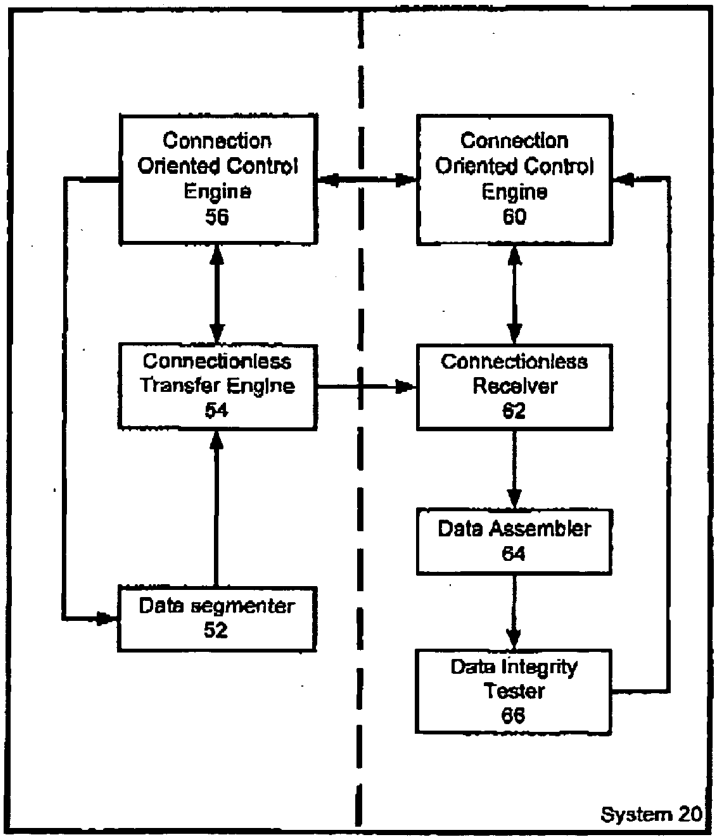
(74) Agents: KINSMAN, L., Anne et al.; Borden Ladner Gervais LLP, 1000-60 Queen Street, Ottawa, Ontario K1P 5Y7 (CA).

(81) Designated States *(national)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CR, CU, CZ, DE, DK, DM, DZ, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, TZ, UA, UG, US, UZ, VN, YU, ZA, ZW.

(84) Designated States *(regional)*: ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZW), Eurasian

*[Continued on next page]*

# (54) Title: METHOD AND SYSTEM FOR SEGMENTED FILE TRANSFER


Connection Oriented Control Engine 56
Connection Oriented Control Engine 60
Connectionless Transfer Engine 54
Connectionless Receiver 62
Data Assembler 64
Data segmenter 52
Data Integrity Tester 66
System 20

(57) Abstract: An architecture for improving the waiting time to download documents from the Internet and a set of methods are disclosed. A methodology is disclosed to improve download time from a number of sites containing component data denoted component based downloading (CBD). The components are downloaded autonomously and reassembled at the client site. A methodology is disclosed for downloading large files using an application denoted file segment transfer protocol (FSTP). FSTP is an application which sends data over UDP and provides its own error control and retransmission. Transfer rates and parameters such as packet size which effect file transfer latency can either be pre-selected or determined dynamically. A second architecture is disclosed which is denoted Distributed File Transfer (DFT). DFT illustrates a more automated approach to file size and server load determination procedures.

WO 01/54370 A2

patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE, TR), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).

**Published:**

— *without international search report and to be republished upon receipt of that report*

*For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.*

# METHOD AND SYSTEM FOR SEGMENTED FILE TRANSFER

## FIELD OF THE INVENTION

The present invention relates generally to data transfer via a telecommunication system. More particularly, the present invention relates to a method and system for segmented file transfer over a communications network, such as the Internet.

## BACKGROUND OF THE INVENTION

The Internet has seen tremendous growth within the last few years. One of the main reasons for its popularity is quick and easy access to a wide variety of data from remote locations. The size of the data varies extensively. It might be a small web page, a relatively large multimedia file, a large video file or an extremely large archive (e.g., a new version of Netscape). In spite of these capabilities, Internet users tend to complain about the time that they waste sitting behind their computers, waiting for data. In other words, for a typical Internet user, latency is a big concern. Latency comes from different sources. For instance, if a server is overloaded or has a slow disk, it imposes a considerable delay before processing a request. Or, if a user's computer does not quickly parse the packets being received and process them, further delay is added.

The latency caused by a server or client can be largely eliminated by using a more powerful computer, more memory, or a faster disk. The main portion of the latency perceived by an Internet user is caused by the network itself. Some sources of this delay are intrinsic to the network infrastructure, namely, propagation and transmission delays. Fortunately, many Internet users presently have access to a reasonably high-bandwidth Internet connection. Other sources of delay are due to network congestion and router delays (the buffering & processing time involved for routing each packet). Yet another source of latency is due to the design/implementation of the Internet protocols themselves; for instance, the delay incurred for retransmission of a lost packet, while using a sliding-window flow control mechanism. These protocols are designed to best match particular network characteristics with the type or size of the data to be transmitted. Therefore, with the evolving nature of the Internet, the protocols are modified occasionally in order to optimize performance.

Many modifications to the existing protocols, mostly of HyperText Transfer Protocol (HTTP), have been proposed in the literature to reduce latency. Some of them have been already implemented on the Internet. These include pre-fetching web pages the user is likely to access next, while browsing the current displayed page; using multiple Transmission Control Protocol (TCP) connections to the server, currently used by web browsers that comply with HTTP1.1; and using mirror servers (i.e.; spreading the workload among a cluster of servers rather than a single machine handling the HTTP requests).

Current file transfer protocols on the Internet are primarily based on using TCP running on the Internet Protocol (IP). TCP was designed for operation over IP, and thus it does not assume that the underlying network service is reliable. TCP provides a connection oriented service, which requires that the two parties communicating with each other progress through the phases of establishing a connection, transferring data, and releasing the connection. To achieve error free connections, TCP implements a selective Automatic Repeat reQuest (ARQ) technique, whereby repeated packets are requested when needed, through the use of a positive acknowledgement system. Additionally TCP implements congestion control by identifying congestion through packet loss monitoring. TCP then minimizes congestion by controlling the rate at which information enters the network through a congestion window.

TCP was defined in the early eighties when the transmission medium was a communication bottleneck. (There have been numerous modifications to enhance the performance of TCP, such as "selective acknowledgement", "window scale option", "round-trip time measurement", and "protection against wrapped sequence numbers".) Today, the emerging use of high-speed networks, fiber optic links, and powerful routers, has dramatically reduced the number of corrupted or lost packets in the Internet. This has made the existing transfer protocol redundant and slow in some cases, due to their assumption of a very unreliable network layer. TCP is the most widely used transport protocol on the Internet, although somewhat mismatched with the speed and reliability of gigabit class networks. This mismatch is mainly due to TCP's use of a sliding-window mechanism and a slow start procedure to ensure a reliable virtual connection. Slowness of TCP has the motivation for designing faster application protocols. This has been achieved as a trade off between speed and loss of data integrity -i.e., no guarantse-that the user. receives all the packets. For example, TFTP (Trivial File Transfer Protocol) uses User Datagram Protocol (UDP) as its transport-level service to move files around with no reliability guarantee. Though TFTP is acceptable on a small LAN, or subnet, where data can be reliably delivered it is not feasible in the internet as a whole, where there is substantial reliability on a per packet basis but there is still no guarantee that the data will arrive in order.

File transfer across a network that utilizes TCP as the transport layer protocol is carried out after a bi-directional connection is made between the two parties. The TCP connection is considered reliable because TCP guarantees that the data is provided to the application layer in order and without error. TCP gives each connection a randomly assigned long (32 bit) sequence of numbers as an identifier to allow a terminal to distinguish between multiple TCP sessions. In essence each TCP connection, when it is created, is assigned one number that it is identified by. Because there are a large number of possible identifiers the probability that two sessions on a single system will have the same identifier is small. To transfer data, TCP provides a sliding window mechanism to track packets, and to assist in reassembling them in correct order for the application layer. To transmit data, TCP requires the sender to fragment the data into segments, which are then encapsulated in a Transport Protocol Data Unit (TPDU). The TPDU has a header containing information about the source, the destination, the session ID, and several other factors. The TPDU is transmitted either upon receipt of a push command from the application layer, or when a counter expires. A window, the sliding window, is defined and a number of TPDUs are transmitted. The receiver sends an acknowledgement (ACK) for each TPDU that is received without error. When the earliest TPDU in the window is acknowledged then the window slides to allow another TPDU to be transmitted. If an ACK is not received for a TPDU within a set time, or before a TPDU near the end of the window is transmitted, then the TPDU is resent. Though this does provide for the arrival of all the TPDUs, it also introduces the unnecessary retransmission of TPDUs. When a TPDU is received, but the ACK is lost in transmission a new copy of the TPDU is sent to the receiver, which them discards the TPDU and sends another ACK, which results in unnecessary traffic. Upon receiving all the TPDUs that constitute the data being transmitted, TCP provides the completed data to the application layer. This allows TCP to ensure that the data has arrived, and is presented in order without error. TCP was designed to be robust in an unreliable environment, and as a result has a high overhead in simple file transfers. One of the most widely used file transfer application programs on the Internet is File Transfer Protocol (FTP). However, because FTP relies on TCP transport layer protocols, it is inefficient in moving large data sets in a timely manner due to the overhead required to implement and process acknowledgements and error correction.

Different approaches have been taken to overcome the inefficiencies of FTP for file transfer. For example, Sun Microsystems, Inc. has designed a web network file system (WebNFS™) for sharing files over the Internet. WebNFS™ is an update of Sun's open standard NFS ™ protocol, which allows client systems to mount partitions located on a server.. Transfer of files can then be accomplished through the use of standard copy commands. Additionally both WebNFS ™ and NFS ™ allow transparent access to the data on the shared partition, whereby a user can execute applications from the shared partition, and otherwise act as if that partition was a part of their physical machine. NFS™ is typically only used on local networks, due to the requirement of reliability and high overhead introduced on larger networks. WebNFS™ still uses TCP because of the generally unreliable nature of the Internet, compared to local networks.

In addition there are several proprietary protocols for file transfer that are based on the UDP where the file transfer application is responsible for the reliable transfer of data. UDP itself is a connectionless unreliable transport protocol. UDP relies upon the either network, or the application layer to ensure that messages are transmitted correctly. UDP does have a header, but it is minimal in nature to reduce overhead. UDP was originally designed for short communications where the overhead of establishing a TCP connection would be considered prohibitive. One common use for UDP is in requests to Domain Name Servers (DNS). A DNS query typically provides an alphanumeric string, and the response is a numeric address associated with the alphanumeric string. These requests are short and bursty in nature and thus establishing a connection-oriented session is not needed. If the user does not receive a response to the DNS query the query can be re-issued. UDP relies upon a reliable network infrastructure, whereas TCP assumes an unreliable network. Because traditionally UDP requests were only sent over a short topological distance the network was considered reliable.

Several proprietary transfer techniques have been developed using UDP, the most prominent of them being developed for use in RealAudio™ and RealVideo™ applications.

In these applications a server segments the data into TPDUs based on UDP, and transmits them to the user. The packets arrive out of order due to the unreliable nature of the Internet. To accommodate the out of order nature of the data, a buffer is employed by the client to allow the data be reconstructed in order. Occasionally packets do not arrive, or do not arrive before they are needed, and the missing packet is skipped in the presentation of data. The listener, or viewer, does not notice the missing data, and thus to the end user the system has low overhead, and through the use of a buffer it ensures that the data is presented in order. Unfortunately, these applications do not have a method of ensuring that a packet is delivered, making them as unacceptable in applications where data integrity is important.

It is also well known that file transfer mechanisms can be distributed over a number of servers. Typically this is done through the use of mirrored servers. To ensure data availability, in the face of either network outage, high demand, or other such problem, many servers on the Internet contain copies of the same data set. Because the standard FTP application has the ability to resume a transfer that had been previously terminated it is possible, if a group of mirror servers are known, to have an FTP client concurrently download different segments of the same file from different servers, to reduce the bandwidth limitations incurred on a busy server. This segmenting technique is purely dependent upon the client, and the client performs the re-assembly of the transferred file. Effective architectures for efficiently transferring large data sets however require network and server load performance metrics to be effective.

Typically, a number of servers can be joined together in a load balancing arrangement, so that in the event of heavy demand the demand is spread among a number of servers. In common load balancing arrangements all client requests are routed to a load-balancing server which determines which server the client request should be routed to. There are a number of techniques that can be applied, but the most efficient is a load monitoring technique, where the load balancing server directs the client request to the server with the least load. U.S. Patent No. 6,078,960 to Ballard, entitled "Client-Side Load-Balancing in Client Server Network", discloses a method of retrieving a file from a plurality of servers without interacting with a load balancing server. The problem of a load-balancing server is that since all requests are routed through a central server there is a single point of failure at the load-balancing server. Ballard discloses a method where the client itself determines the server with the lowest load and directs its transfer to that server. This method requires the use of a local load-balancing list that contains information on servers, including the share of the load that the server should carry, and optionally contains information about the load carried by each server. This information is obtained from one of the servers. The client then uses the list to select a server with the most excess capacity, and begins the transfer with that server. There are protocols available for events such as the unavailability of a selected server that are also disclosed.

U.S. Patent No. 6,085,251 to Fabozzi III, entitled "Implementing a Parallel File Transfer Protocol", discloses a method of transferring a file to a client from a server. The method disclosed by Fabozzi requires segmenting the file, creating a log to aid in the reconstruction of the segmented file, transferring the log to the client, and then having the client request each of the segments in different file transfer sessions and reconstruct the file upon receipt of all the segments. This method seeks to overcome the problems that may arise with a sliding window, by creating multiple sliding windows, and letting a few sliding windows do the transfer. This method of multiply accessing a file does introduce efficiencies, and is often limited either by bandwidth restrictions at a single server or by restrictions that servers impose on multiple logins from a single client. Additionally there is a point of diminishing returns, where the overhead of the different TCP sessions becomes more of a burden on the transfer than the additional session provides in throughput.

It is, therefore, desirable to provide a method and system for transferring files, particularly large files, over the Internet that provide reliability and speed without high overhead.

## SUMMARY OF THE INVENTION

It is an object of the present invention to obviate or mitigate at least one disadvantage of previous systems and methods for file transfer. It is a particular object of the present invention to improve the performance of Internet-based applications that require moving data from one or more servers over the Internet, moving data between clients and servers, moving data between peers in a peer-to-peer network, and moving data between cache appliances, in a single or multiple server environment.

In a first aspect, the present invention provides a method for transmitting data to a client in a distributed network environment. This transmission method begins by establishing a control channel to a client. This control channel optionally employs a connection-oriented protocol, such as TCP. The data to be transmitted to the client is then segmented into data segments, and the data segments are transmitted to the client using a connectionless protocol, such as UDP. In a presently preferred embodiment, a sequence number is attached to each data segment prior to transmission. Repeat requests, requesting retransmission of any erred data segments are received over the control channel. To improve performance, an inter-packet transmission delay can also be included.

In a further aspect, there is provided a method for receiving a data file in a distributed network environment. The reception method begins by establishing a control channel to a server. This control channel optionally employs a connection-oriented protocol, such as TCP. Data file segments, transmitted from the server, are received using a connectionless protocol, such as UDP. A data file is assembled from the received data file segments, typically by ordering the segments by their sequence numbers, or through supplemental control information. If the assembled data file contains erred segments, a request for transmission of the erred segments is made over the control channel.

In yet a further aspect there is provided a method for transmitting a data file to a client in a distributed network environment. The method begins by establishing control channels from a plurality of servers to a client. These control channels optionally employ a connection-oriented protocol, such as TCP. The data file is divided into components on the plurality of servers, and at each of the plurality of servers, the components are segmented into data segments to be transmitted to the client. The data segments are then transmitted to the client, in parallel, using a connectionless protocol, such as UDP. A repeat request, requesting retransmission of erred data segments, can be received over any of the control channels. Load balancing can be implemented to improve performance.

In yet another aspect, there is provided a file transfer system for transmitting a data file to a client in a distributed network environment. The file transfer system includes a data segmenter for dividing a data file into components on a plurality of servers and for segmenting the components into data segments to be transmitted to a client. A connectionless transfer engine, in communication with the data segmenter, transmits, in parallel, the data segments to the client using a connectionless protocol. Finally, control engine, in communication with the connectionless transfer engine, can establish control channels from the plurality of servers to the client, and can receive, over at least one of the control channels, a repeat request requesting retransmission of erred data segments. The control engine optionally includes a connection oriented control engine, which optionally employs a connection-oriented protocol to establish a control channel.

At the reception side, there is also provided a file transfer system for receiving a data file in a distributed network environment. This receiving system includes a control engine that establishes a control channel to a server. The control engine optionally includes a connection oriented control engine, which optionally employs a connection-oriented protocol to establish a control channel. A connectionless receiver, in communication with the connection-oriented control engine, receives data file segments transmitted from the server using a connectionless protocol. While a data assembler, in communication with the connectionless receiver, assembles a data file from the received data file segments. A data integrity tester, in communication with the data assembler, can then determine if the assembled data file includes erred segments, and can relay to the connection-oriented control engine a request for transmission of the erred segments.

Other aspects and features of the present invention will become apparent to those ordinarily skilled in the art upon review of the following description of specific embodiments of the invention in conjunction with the accompanying figures.

## BRIEF DESCRIPTION OF THE DRAWINGS

Embodiments of the present invention will now be described, by way of example only, with reference to the attached figures, wherein:

Fig. 1 is a block diagram of a communications system according to the present invention;

Fig. 2 is a block diagram of the logical components of a file transfer system according to the present invention;

Fig. 3 is a flow chart of the component-based download method of the present invention;

Fig. 4 is a flow chart of the file segment transfer protocol method of the present invention;

Fig. 5 is a flow chart of the combined component-based download and file segment transfer protocol method of the present invention; and

Fig. 6 is a flow chart of the distributed file transfer method of the present invention.

## DETAILED DESCRIPTION

Generally, the present invention provides a method and system for transferring data files over the Internet, or other distributed communications network. Ideally, the present invention results in improved data transmission performance as viewed from an individual user's perspective. First, a distributed approach for downloading files, referred to herein as Component-Based Download (CBD), is introduced. Second, an application protocol for more efficient and reliable transfer of large data sets, referred to herein as File Segment Transfer Protocol (FSTP), is described. Third, the combination of CBD and FSTP, referred to herein as CBD-FSTP, is described. Fourth, a file transfer architecture, referred to herein as Distributed File Transfer (DFT) architecture, is described. DFT uses standard file transfer protocols to adaptively download segments of replicated files stored on distributed servers. Finally, the results of several simulations and tests are presented to demonstrate the advantages of the proposed method and system.

The following description will focus on the presently preferred embodiment of the present invention, which is operative in an Internet-connected environment, including, for example, machines operating under a Microsoft® Windows or UNIX® environment and connected to an open network, such as the Internet. The present invention is not, however, limited to any particular one environment. Instead, as will be understood by those of skill in the art, the system and methods described herein can be advantageously applied to a variety of system and application software, operating on a variety of platforms, including the Macintosh® operating system, and the like, and can apply to systems implemented on any wide area network (WAN), or local area network (LAN).

Referring to **Fig. 1**, a block diagram of the system of the present invention is shown, and generally referenced at **20**. System **20** generally consists of client(s) **24**, servers **28**, **32**, and **34**, interconnected via the Internet **36**. Clients **24** are typically embodied on a general purpose computer, such as an IBM PC-compatible personal computer, operating as stand-alone machines or interconnected on a LAN. For example, in a LAN based environment, client can consist of a central processor, a main memory, such as conventional random access memory (RAM), an input/output controller, a keyboard, a pointing device, such as a mouse, track ball, track pad, pen device or the like, a display or screen device, a mass storage device, such as a hard drive, floppy drive, optical disk, flash memory or the like, a network interface card or controller, such as an Ethernet card, and a modem. Client **24** communicates with other systems via the network interface card and/or modem. A variety of optional input/output devices (not illustrated) can also be attached to client **24**, including printers, slide output devices, plotters, etc. Typically, a computer software system is provided for directing the operation of client **24**. The software system is stored in memory, and on the mass storage device, and generally includes a kernel, or operating system (OS), and a windows shell. The OS and windows shell can, for example, be provided by Microsoft® Windows 98, Microsoft® Windows NT, IBM OS/2®, Macintosh® OS, Linux, or other similar application. One or more application programs, such as client application software, can be loaded for execution by client **24**. In a presently preferred embodiment, client application software includes a web browser, such as Microsoft Internet Explorer™ or Netscape Navigator™ browser software that communicates with the Internet through a communication layer or driver, such as a Winsock driver. The software system typically further includes a user interface, preferably a graphical user interface (GUI), for receiving and displaying user inputs, commands, and data and outputs resulting from operation of the operating system and the application programs.

Servers **28**, **32** and **34** can each consist of one or more conventional computers having a Pentium™ class, or better, central processing unit, such as manufactured by Intel™ Corporation, Santa Clara, CA, or other similar processing unit. Each of the servers can be provided with peripheral components and software components as described above with respect to client **24**. Servers **28**, **32** and **34** can be attached to large mass storage devices, such as a RAID storage device. Those of skill in the art will understand that servers **28**, **32** and **34** can be distributed in various locales and can communicate with each other over dedicated lines, or over Internet **36**.

The logical components of the file transfer system **20** are shown in **Fig. 2**. The logical components are embodied in appropriate application software, as is well known to those of skill in the art. On the transmitting side, file transfer system **20** consists of a data segmenter **52** for segmenting a data file into data segments to be transmitted to a client. A connectionless transfer engine **54** is in communication with the data segmenter **52**. The connectionless transfer engine **54** transmits the data segments to the client using a connectionless protocol, such as UDP. A connection-oriented control engine **56**, in communication with the connectionless transfer engine **54**, establishes control channel(s) to the client using a connection-oriented protocol, such as TCP, and can receive repeat transmission requests for erred data segments. As used herein, an erred data segment means missing/lost or corrupt data segments.

On the receiver side, system **20** consists of a further connection-oriented control engine **60** that communicates with the connection-oriented control engine **56** to establish a control channel to the server(s) using a connection-oriented protocol. A connectionless receiver **62**, in communication with the connection-oriented control engine **60** and the connectionless transfer engine **54**, receives data file segments transmitted from the server using a connectionless protocol. A data assembler **64**, in communication with the connectionless receiver **62**, assembles a data file from the received data file segments. A data integrity tester **66**, in communication with the data assembler **64**, determines if the assembled data file includes erred segments, and relays a request for retransmission to the connection-oriented control engine **60** which can send the request to the connection-oriented control engine **56**.

As described below, the control engine of the system **20** need not be connection oriented. It is possible to implement a system, within the scope of the present invention, that uses a connectionless protocol to establish a control channel. This control channel need only be bi-directional to facilitate communication. Connection oriented channels introduce benefits, such as guaranteed arrival of control information, that though beneficial are not essential to the operation of the present invention.

The operation of system **20** and the method of the present invention will now be described with reference to **Figs. 1 - 6**. The following description assumes that the client **24** is connected to Internet **36** and has access to the servers **28**, **32** and **34** for the purposes of file download.

Referring to **Fig. 3**, using CBD, a file to be downloaded is split into components, or clusters, at step **200**. The number of components is determined by the number of available servers **28, 32** and **34**, while the size of the components can be determined by the relative connection speeds, server speeds, or other suitable factors. Ideally, the component sizes are chosen such that each component can be downloaded to the client **24** in approximately the same time. Each component is then placed on a different server at step **202**. When a user, situated at client **24**, attempts to download the file, parallel, or concurrent, connections are established, at step **204**, between the client **24** and the servers **28, 32** and **34** maintaining the file components. All the components are then downloaded simultaneously at step **206**. At the client side, after the download is complete, the components on the client machine are appended to each other at step **208**, thereby reconstituting the original file.

In other words, with CBD, a file is first divided into several components or clusters and put on different servers. To initiate the download of the file cluster, parallel FTP sessions are established between the user and the respective servers and files are downloaded to the user's side in a parallel fashion. On the user's side, each FTP session is terminated after the respective component is completely downloaded. This leads to a faster transmission rate, and thus a considerably shorter latency perceived by the user.

CBD is inherently fault tolerant. There are occasions in which a server may go down, or a router starts discarding the packets when buffers are full. This means file components may no longer be accessible if they reside on that part of the sub-network. This problem can be addressed by resorting to a replication scheme. Depending on the nature of the file cluster, and the network's environment, several simple replication schemes can be implemented. One is to put the whole file cluster on a backup server. The other is to divide the file into replicated components and spread them among the servers.

In order to substantiate the assumption that CBD is faster than the conventional approach, a series of preliminary experimental measurements were carried out. In these experiments the CBD paradigm used FTP as the file transfer protocol (FTP was chosen, because it is the de facto protocol for downloading files through the Internet), the respective latencies were measured, and compared to the latencies while using the conventional method. To do that, several files, located on different FTP servers, were downloaded by establishing simultaneous FTP sessions (i.e.: concurrently). Then, the same files were downloaded, by opening FTP sessions with those servers one after another (i.e.: sequentially). Measurements were performed and the results are discussed in detail below. Extensive experimental results were collected over campus type LANs, modems, and wireless LANs supporting the wireless protocol 802.11. Three different FTP servers at remote locations were chosen: ftp.microsoft.com in USA, ftp.unix.hensa.ac.uk in England, and ftp.dti.ad.jp at Japan. On each server, relatively large files were selected for download (2, 2 & 4.7 Mbytes, respectively). A UNIX/SUN workstation connected to a 100BaseT LAN was used as the client machine, housed at the University of Manitoba. The time it takes to download the files to the local machine was measured in two ways: (1) The three files were consecutively downloaded, and the latencies incurred in downloading each file was measured. The latencies were added, which gave us the total latency. (2) The same three files were downloaded, this time by establishing three simultaneous FTP sessions with those servers. We measured the time interval between initiation of the first FTP session and termination of the last session. We ran the FTP sessions simultaneously by spawning parallel threads for each one. Intuitively, this time is the latency perceived by the user while using the CBD approach. In order to be able to repeat the experiment for a reasonable number of times, a Java application was created to automate the process of opening the FTP sessions, measuring and collecting the latencies. In the experiments, several contingencies that might occur in a real-life situation were covered. To compensate for the effect of variations of the traffic load on the network, experiments were repeated at various hours of the day on different days of the week. We repeated our experiments 70 times to get a reliable result. According to our measurements, the average time for downloading files through the component-based approach was 13.5 seconds, as opposed to conventional approach delay with an average of 25.8 seconds. These results clearly illustrated the potential to improve the performance of a network by using CBD for large files.

In a second test, the total latencies for each scenario were calculated as follows: In concurrent download, the perceived latency is equal to the greatest value of latencies measured for each download. In sequential download, the perceived latency is equal to the sum of latencies measured for each download. The experiments were performed on 4 different environments: 100BaseT LAN, 10BaseT LAN, 802.11 wireless LAN, and with a dialup modem. First, three files with total size of 4.5 megabytes; second, three larger files with total size of 26 megabytes; and third, five files with total size of 9 megabytes were downloaded.

The results clearly illustrate the potential to improve the performance of a network by using CBD for large files. Latency can be reduced to less than 50% compared to when downloading in conventional way. These results are summarized in **Table 1**.

**Table 1:** Relative Latencies for CBD compared to traditional FTP

| Size/No. of Files | Environment | | | |
|---|---|---|---|---|
| | 100BaseT | 10BaseT | 802.11 | Dial up modem |
| 3 Files (4.5 Meg) | 0.48 | 0.47 | 0.46 | 1.06 |
| 3 Files (26 Meg) | 0.45 | 0.39 | 0.74 | - |
| 5 Files (9 Meg) | 0.34 | - | 0.32 | - |

In this table relative latencies (the latency for CBD method divided by the latency for sequential download) are presented. This method can be extended using various numbers of TCP connections, larger files, and various access methods to determine an optimal range of TCP connections, which gives us the best latency. Opening more TCP sockets enhances the latency, but this would also introduce more network and processing overhead.

Referring to **Fig. 4**, to further improve the latency, a novel application-level protocol, FSTP, is shown. In this protocol a TCP socket connection is established at step **300**. The TCP socket connection forms the control channel and is used for sending commands. Concurrently with step **300**, or separately, one or more UDP socket connections are established at step **302** to enable data transmission. By contrast, in FTP, which is the current dominant protocol for transferring data, TCP sockets are used for both the control and data channels. At step **304**, the desired data file is transmitted as data units, such as datagrams or packets of predetermined size, as is well known to those of skill in the art. Since UDP does not provide a reliable connection, the integrity of the data transfer is checked in the application level at step **306**. FSTP takes advantage of the more reliable network infrastructure currently available in the Internet, as compared to when TCP was first introduced. Preliminary tests show that FSTP transmits data much faster than FTP. Experimental results have been obtained for campus LANs, wireless LANs, and traditional telephone modems and cable modems. In the worst case, FSTP performs no worse that FTP, and in the best cases to date with experiments over broadband networks typical file transfer speed-up improvements are on the order of 500-800%.

To better understand the FSTP, the network connections used for an FTP session are explained and compared to the transport layer connections used in the FSTP protocol. FTP uses two TCP connections, one for exchanging command/control packets, and the other one for the data itself. Basically, FTP protocol is not concerned about retrieving the missing/corrupted packets. The job of providing a reliable network connection is delegated to the transfer layer protocol, i.e., TCP. TCP ensures the integrity of the data by checking the incoming packets, and asking the other side for retransmission of the erroneous/missing packets.

FSTP operates significantly differently from FTP. Although there is still a TCP network connection to send command/requests to the server, all of the data is transferred via UDP packets. UDP here affords simple access to the Internet Protocol (IP) layer. Sending data over UDP does not bind us to restrained performance of a TCP connection's sliding window flow control. Another benefit in using UDP packets is that there is less processing overhead compared to a TCP connection. However, since UDP only provides a datagram service, the necessary functionality for data consistency is provided in another level. This task is performed in the application level (i.e., by FSTP itself). In a presently preferred embodiment, FSTP attaches a unique sequence or tag number to each packet. On the client side, after FSTP receives all of the data packets in a stream, it looks for missing/corrupted packets. The missing packets may be logged during the transfer or determined after the transfer is completed. If found, the FSTP client sends a request for retransmission of the missing packets back to the FSTP server. The server retrieves the missing parts of the file from its local disk or memory, and sends them again to the client. This loop continues until the data is completely transferred to the user.

Clearly, transmitting packets at the maximum possible speed can result in many of the packets getting lost in transit due to a smaller maintainable bandwidth over certain portions of the Internet and/or an inability of the client to process the incoming datagrams. To maintain a reasonably small percentage of packet loss, an inter-packet transmission delay can be added. Adding this delay results in a greater packet reception success rate (with respect to the number of packets transmitted), and results in fewer required retransmissions. Because the examination of received packets, generation of a retransmission requests, and processing of these request by the server takes of time, the perceived latency to the user will actually be less with the use of the appropriate inter-packet transmission delay for a pre- chosen packet size. Adaptive inter-packet transmission timing scheme will also work to minimize excessive network traffic.

To optimize throughput between the client and the server, FSTP can, for example, conduct a brief test where the server attempts to flood its connection with UDP packets destined for the client for a short time interval (under 1 second). The client counts the number of packets it received out of the total and, with the noted transmission time from the server, calculates an appropriate delay. For example, if the server sends 1000 packets in one second and the clients receives 100, we can speculate that if we transmit a packet every 10mS, we should be able to maintain a high packet reception success.

The presently preferred initialization steps a FSTP file transmission are: FSTP client opens up a TCP connection with the server for exchanging commands. The client receives a list of files and their respective sizes. The packet size is chosen by the user and this selection is forwarded to the server. The server begins the transmission timing test by sending UDP packets of the specified size to the client. After the test has completed, the server sends the client a message indicating the total transmission time for the test and the number of packets transmitted during the test. The client calculates an appropriate inter-packet transmission time and sets this transmission delay on the server. The client sends a "SEND" command by specifying the name of the file to be retrieved, taking note of the file's size from the previously acquired file listing.

Both the client and the server can calculate the number of bytes necessary for the packets' sequence numbers, or tags. It is necessary for the client to calculate this as well, so it will properly handle the format of the received packets without the server having to explicitly send a description of the format. This is possible because the client and server use the same program library to handle tag numbers. The server sends the file as a steam of UDP packets whose headers contain the file name and tag made of bytes to indicate the position of the data in the file. Other methods with less overhead can be used, such as after the stream of UDP packets has ended, the client generates a list of missing/corrupted packets and sends them in a retransmission request to the server; the server retransmits the requested packets in the same format as the original transmission; and this retransmission continues until all of the packets are received. To pinpoint the corrupted packets, the checksum capability of UDP protocol can be used. The current, experimental system attempts to establish an average maintainable data rate in packets per second (for a chosen packet size) and transmits the data from a single server to the client.

An alternative method would be for the server to adapt its rate and packet size in response to the client. The packet size is easily adjusted and is often optimal if near the maximum segment size of any of the intervening networks a packet would traverse from source to destination. For example, based on all experiments to date a packet size of 1500 bytes has been found to be near optimal. In addition the inter-packet transmission delay can be adapted using a simple feedback control increasing or decreasing the rate depending on the rate of packets received. If for example, if only 80% of the packets were being delivered the server can throttle back its transmission of packets into the network until a desirable reception rate is achieved.

To test the performance of FSTP and compare it with that of FTP, some preliminary measurements were performed. Three types of machines and network connections were chosen. The same file was downloaded, using FTP and then by FSTP. The time taken for each download was recorded and compared. From the results of the tests, when a 1500-byte IP packet size is used with FSTP transfers (as the optimum packet size), the transfer time range from approximately the same as FTP times to nine times faster. This is most likely because the FSTP transfers do not depend on acknowledgement for each packet, and are therefore not bound by round trip propagation delay over the network. The other fact derived from the test is that FSTP does not improve the latency when used in a low-bandwidth network connection (e.g., dialup modem). This is not surprising, because FSTP is basically designed based on the assumption of exploiting the large available bandwidth (which is exactly the shortcoming of FTP that uses TCP for transferring data). So, FSTP performs best, compared to FTP, when the two machines (client and server) are connected via a fast Internet connection, with a relatively high RTT (round trip time). **Table 2** shows the average transfer times for FTP and FSTP measured in seconds.

**Table 2:** Latencies for FTP and FSTP (seconds)

| File Size | Environment(packet size) | | | |
|---|---|---|---|---|
| | FTP | FSTP(500) | FSTP(1000) | FSTP(1500) |
| (15 Meg) | 192 | 35 | 21 | 19 |

A file was transferred from a remote machine (located at the University of Victoria) to a local machine (located at the University of Manitoba). The two machines were Sun Sparc™ workstations, running Solaris™ 5.6 and 5.7. They were connected via a high-speed connection. The measurements were repeated 5 times during a weekday afternoon, when the machines were busy running other processes and the Internet was being used by business and academic users. Three different packet sizes were used, 500, 1000, and 1500 bytes.

The method disclosed in the previous example used a TCP connection for the control channel, due to the inherent reliability of the TCP channel. Though this is advantageous in many circumstances it should be noted that the control channel need only be a bi-directional channel and does not absolutely require the benefits of a TCP channel. One embodiment of the present invention uses a connectionless protocol for the control channel and the transfer channel. This embodiment takes advantage of the lack of overhead to minimize traffic in the control channel. As segments arrive at the receiver they are reassembled as in the previous example. Upon receipt of the final segments a list of erred segments is prepared and sent out over a connectionless control channel, preferably using UDP. Upon receipt of the retransmission list the server(s) transfer the erred segments. If the transmitter does not receive one part of the retransmission list, then it will not be sent to the receiver. The receiver does not remove segments from the erred segment list until they have arrived intact. If segments do not arrive correctly, or do not arrive at all, in the second transmission, then they will be re-requested and the process will continue until all the segments have arrived.

Although good results have been achieved using FSTP (up to 900%), still more enhancements can be done on FSTP protocol, without affecting the basic concepts embodied within FSTP.

Referring to Fig. 5, to integrate the two concepts and achieve even a higher speed of data transmission, a distributed version of FSTP, namely CBD-FSTP, is shown. Under this distributed version, file components are stored on distributed servers and downloaded to the client 24 seamlessly over multiple FSTP connections. A file to be downloaded is split into components, or clusters, at step 200, as described above. Each component is then placed on a different server at step 202. When a user attempts to download the file, parallel, or concurrent, FSTP connections are established, at step 404. As described above, a TCP socket connection is established with each server, and UDP socket connections are established for the data transmission. At step 410, the segmented data file is transmitted as datagrams in parallel from each server to the client. At the client side, after this download is complete, the components on the client machine are re-sequenced at step 412 to reconstitute the original file. A check is made for missing and corrupt packets at step 414, and transmission of any erred packets is requested at step 416.

In this instance FSTP is used within the CBD architecture. This is a simple extension and illustrates the use of FSTP with a concurrent download environment. As expected, since file transfer delay is improved with either method, when combined further improvements are multiplicative.

To incorporate FSTP into the CBD paradigm, some modifications are applied to FSTP. The following features/changes are added to FSTP. An additional field is added to the FSTP header. This field represents the file component to which that packet is belonged.

Since in the CBD approach the client establishes parallel/concurrent connections to multiple servers. The client is implemented using threads. Each thread is responsible for maintaining connection (a TCP and UDP connection) with a participating server.

After the client sorts each component, it appends these components sequentially together to create the original file.

Initial tests with CBD-FSTP shows an even faster download time compared to using FSTP. Preliminary test results for CBD-FSTP connections to Victoria BC Canada, Regina SK Canada, and Edmonton AB Canada, from Winnipeg MB Canada, demonstrate the multiplicative speedup of both techniques. These results are illustrated in **Table 3.** These results illustrate the additional benefit that can be achieved using the FSTP protocol within a distributed file transfer application.

**Table 3:** Latency CBD-FSTP (seconds)

| | | Average Latency (sec) | | |
|---|---|---|---|---|
| File Size | Num. of Servers | CBD-FSTP | FSTP | FTP |
| (20 Meg) | 3 | 59 | 85 | 434 |
| (40 Meg) | 4 | 122 | 198 | 872 |

A novel load balancing method, namely DFT, can also be implemented in the system of the present invention. DFT utilizes a intermediate tier process to coordinate the selection of file servers and to maintain server status information. In the implementation of DFT each server has completely replicated data or files. The degree to which a server provides content is based on measurements made by the client. The client actively probes the connection to the server and monitors the file transfer process balancing the load as required during the download. DFT illustrates the role of active monitoring and the decision process involved in concurrently downloading files over the network. DFT is responsible for partitioning the file in terms of server performance and reassembling the file on the client side. DFT can be deployed with using either traditional FTP or the FSTP protocol disclosed here.

The Distributed File Transfer (DFT) architecture can further automate the process of downloading files from a number of servers. A current instance of DFT can allocate server resources according to their availability and utilization. Therefore, we obtain reliability, scalability and efficiency by making better use of all file servers in the Internet.

This approach makes all FTP servers that contain the same files a cluster of servers and distributes load among them.

The design concept of DFT is as follows: In order to switch among servers, a client connects to multiple file servers to download file components simultaneously. This approach like CBD improves the performance of file transfer in general. The client monitors all connections to file servers to detect failure and congestion problems. If some connection is broken, the DFT client has sufficient intelligence or means to decide which server it should switch to and switch to that server immediately and transparently. This is an improvement over CBD in terms of fault tolerance. Within the DFT architecture there is a Load Distributing Server (LDS). The LDS knows all file servers or can search for them so that it can provide a server list to a client. Thus, the LDS coordinates the load distribution among all servers allowing the client to focus on file transfer among a small set of servers. In the current implementation traditional FTP is used although FSTP can be used if the servers are replaced with FSTP servers.

Referring to **Fig. 6**, the working procedure for DFT is as follows: The DFT client sends a request to the LDS at step **500**. The LDS looks for servers that contains the target file. LDS provides the DFT client with a list of servers at step **502**. The DFT client tests the speed and availability of the servers at step **504**, and calculates the segment sizes for each server at step **506**. The DFT client then establishes multiple data links with those servers and downloads different segments from them at step **508**.

DFT achieves performance improvements by more intelligently selecting and monitoring the file transfer process. DFT initially estimates the server speed and network by issuing ping and/or FTP size commands. This establishes the size of the component to be downloaded from each participating server. Once the file transfer is in progress DFT monitors each concurrent connection and reallocates load as required. The present instance and example of DFT disclosed here utilizes FTP for downloading a 47 Mbyte file from 1 to 5 servers. File transfer speedup for this series of experiments resulted in throughput improvements which increased linearly from 600 Kbps to 1400 Kbps as the number of servers ranged from 1 to 5.

Using the method and system of the present invention a higher throughput is achieve simply by modifying the implementation of the existing application protocols to account for the inherent unreliable data transfer that would otherwise be seen transferring files directly over UDP. Higher throughput means faster data transfer, which leads to a shorter perceived latency by the user. This enhancement is achieved with more processing overhead and network cost. Fortunately, this is not a significant burden on new computers with ever-increasing processing power, cheaper memory, and fiber optic Gigabit links.

The above-described aspects of the invention are suited for transferring all types of files, and have been shown to reduce transfer times. These techniques have been shown to work on files of all sizes, though the improvements are most noticeable with larger files.

The above-described embodiments of the present invention are intended to be examples only. Alterations, modifications and variations may be effected to the particular embodiments by those of skill in the art without departing from the scope of the invention, which is defined solely by the claims appended hereto.

What is claimed is:

1. A method for transmitting data to a client in a distributed network environment, comprising:

(i) establishing a control channel;

(ii) segmenting the data into data segments to be transmitted to the client;

(iii) transmitting the data segments to the client using a connectionless protocol; and

(iv) receiving, over the control channel, a repeat request requesting retransmission of erred data segments.

2. The method according to claim 1, wherein the control channel is established using a connection-oriented protcol.

3. The method according to claim 2, wherein the control channel is established using Transmission Control Protocol.

4. The method according to claim 1, wherein the connectionless protocol is User Datagram Protocol.

5. The method according to claim 1, further including attaching a sequence number to each data segment prior to transmission.

6. The method according to claim 1, further including retransmitting the erred data segments using a connectionless protocol.

7. The method according to claim 1, wherein transmitting the data segments includes determining an inter-packet transmission delay.

8. A method for receiving a data file in a distributed network environment, comprising:

(i) establishing a control channel to a server;

(ii) receiving data file segments transmitted from the server using a connectionless protocol;

(iii) assembling a data file from the received data file segments;

(iv) determining if the assembled data file contains erred segments; and

(v) requesting, over the control channel, transmission of the erred segments.

9. The method according to claim 8, wherein the control channel is established using a connection-oriented protocol.

10. The method according to claim 8, wherein assembling the data file includes assembling the data file segments in order of respective associated sequence numbers.

11. The method according to claim 8, wherein the determining if the assembled data file contains erred segments includes performing a checksum.

12. The method according to claim 8, wherein the determining if the assembled data file contains erred segments includes determining a sequence number for one of the expected data file segments is missing.

13. A method for transmitting a data file to a client in a distributed network environment, comprising:

(i) establishing control channels from a plurality of servers to a client using a connection-oriented protocol;

(ii) dividing a data file into components on the plurality of servers;

(iii) at each of the plurality of servers, segmenting the components into data segments to be transmitted to the client;

(iii) transmitting, in parallel, the data segments to the client using a connectionless protocol; and

(iv) receiving, over at least one of the control channels, a repeat request requesting retransmission of erred data segments.

14. The method according to claim 13, wherein the control channels are established using Transmission Control Protocol.

15. The method according to claim 13, wherein the connectionless protocol is User Datagram Protocol.

16. The method according to claim 13, further including attaching a sequence number to each data segment prior to transmission.

17. The method according to claim 13, further including retransmitting the erred data segments using a connectionless protocol.

18. The method according to claim 13, wherein transmitting the data segments includes determining an inter-packet transmission delay.

19. The method according to claim 13, further including balancing a transmission load between the plurality of servers in accordance with their respective performance metrics.

20. A file transfer system for transmitting a data file to a client in a distributed network environment, comprising:

a data segmenter for dividing a data file into components on a plurality of servers, and, at each of the plurality of servers, segmenting the components into data segments to be transmitted to a client;

a connectionless transfer engine, in communication with the data segmenter, for transmitting, in parallel, the data segments to the client using a connectionless protocol.; and

a control engine, in communication with the connectionless transfer engine, for establishing control channels from the plurality of servers to the client, and for receiving, over at least one of the control channels, a repeat request requesting retransmission of erred data segments.

21. A file transfer system according to claim 20, wherein the control engine includes a connection-oriented control engine.

22. A file transfer system for receiving a data file in a distributed network environment, comprising:

a control engine for establishing a control channel to a server;

a connectionless receiver, in communication with the connection-oriented control engine, for receiving data file segments transmitted from the server using a connectionless protocol;

a data assembler, in communication with the connectionless receiver, for assembling a data file from the received data file segments;

a data integrity tester, in communication with the data assembler, for determining if the assembled data file includes erred segments, for relaying to the connection-oriented control engine a request for transmission of the erred segments.

23. A file transfer system according to claim 22, wherein the control engine includes a connection-oriented control engine.

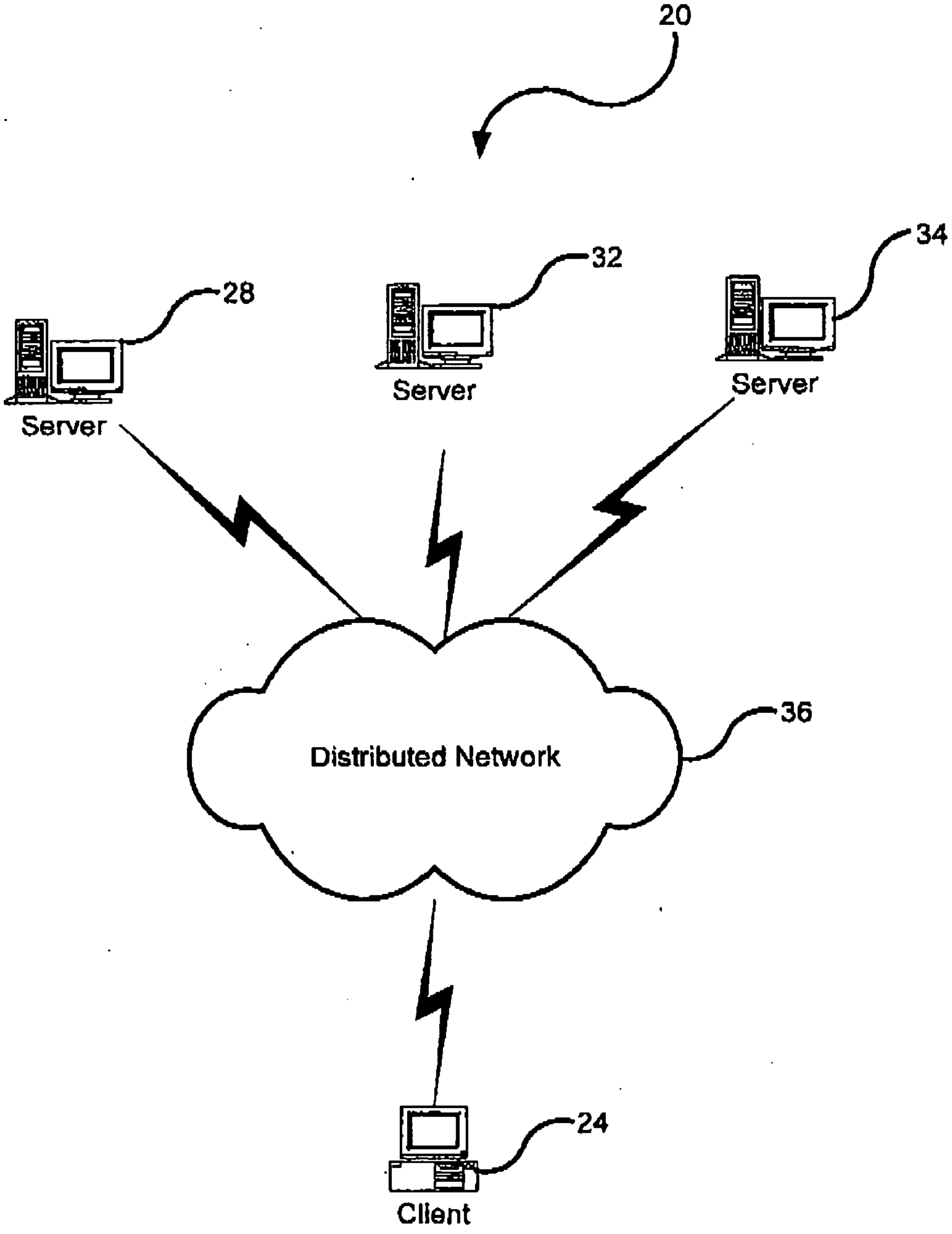

20
34
32
28
Server
Server
Server
36
Distributed Network
24
Client

Figure 1

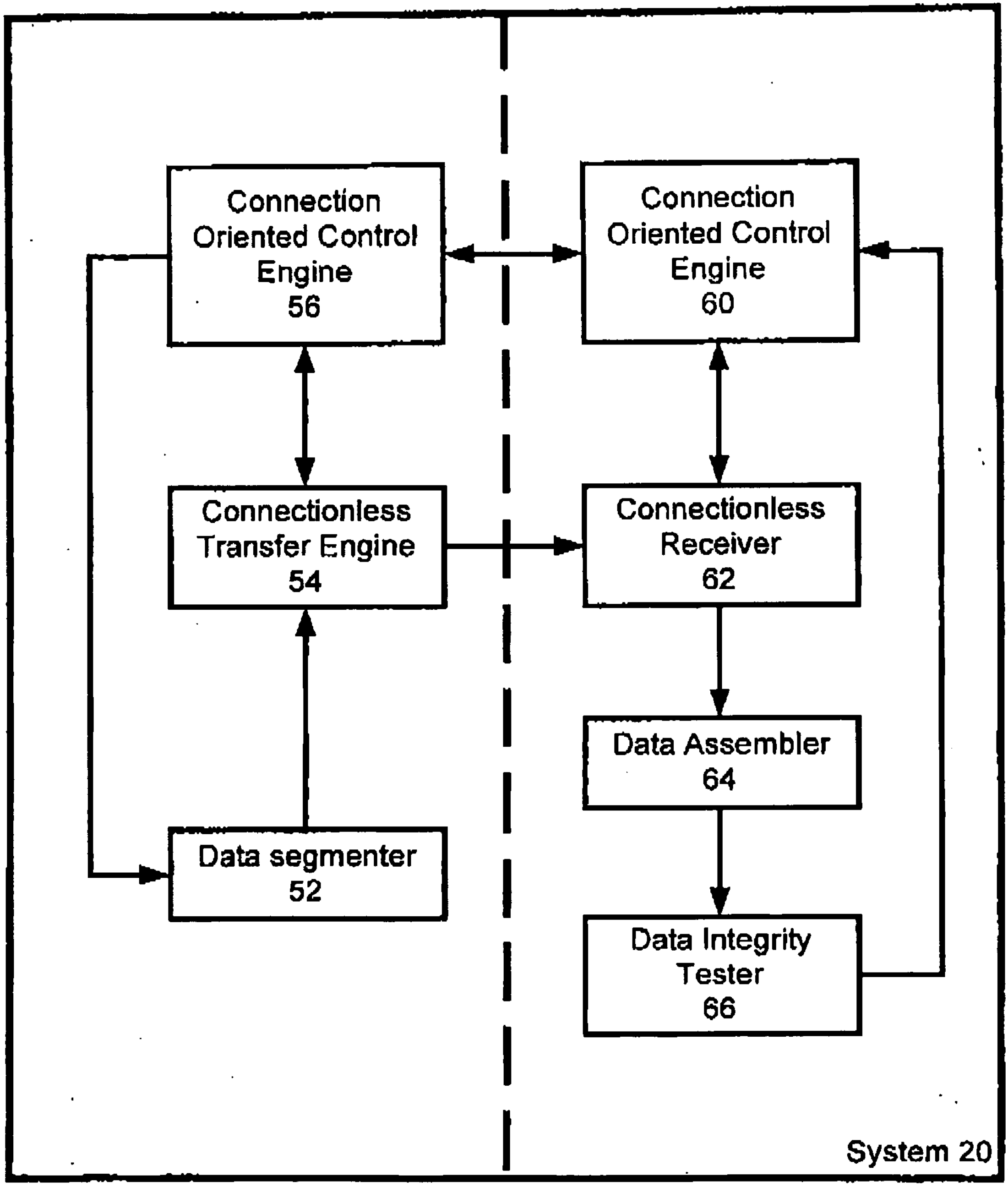

Connection Oriented Control Engine 56
Connection Oriented Control Engine 60
Connectionless Transfer Engine 54
Connectionless Receiver 62
Data Assembler 64
Data segmenter 52
Data Integrity Tester 66
System 20

Figure 2

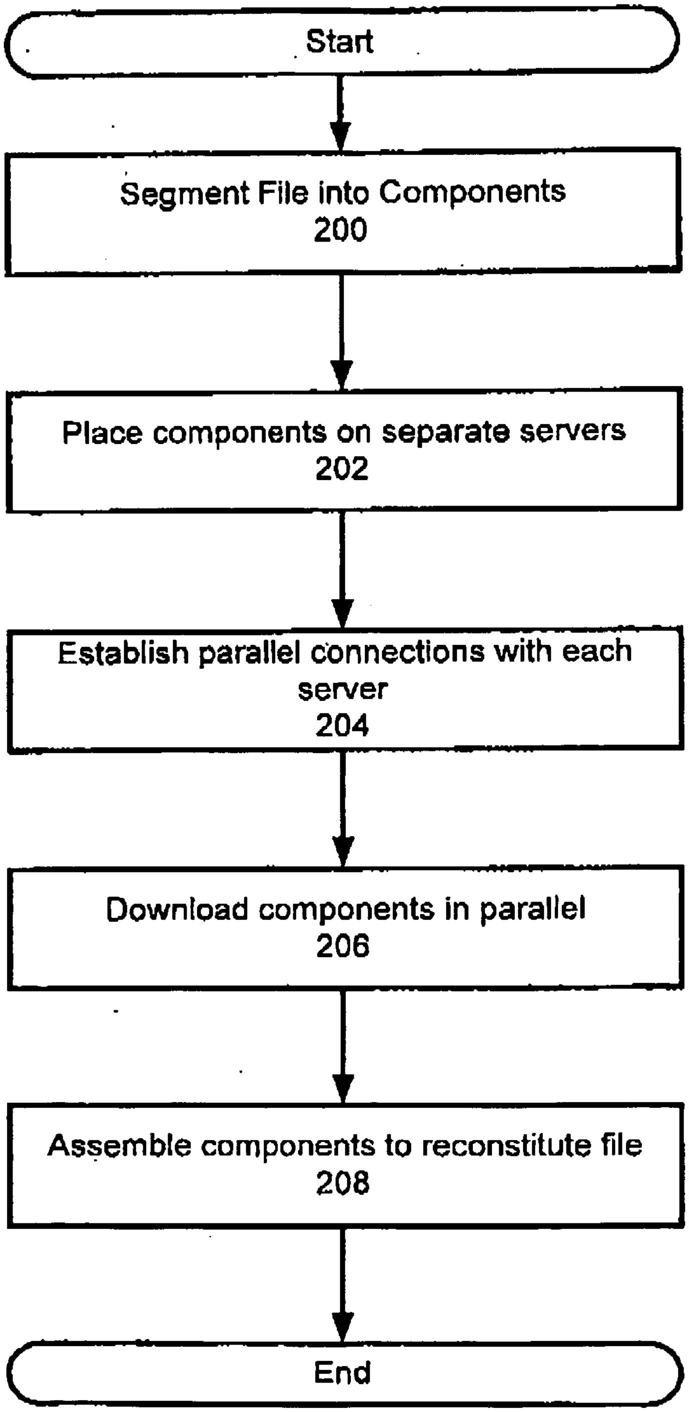

Start
Segment File into Components
200
Place components on separate servers
202
Establish parallel connections with each server
204
Download components in parallel
206
Assemble components to reconstitute file
208
End

Figure 3

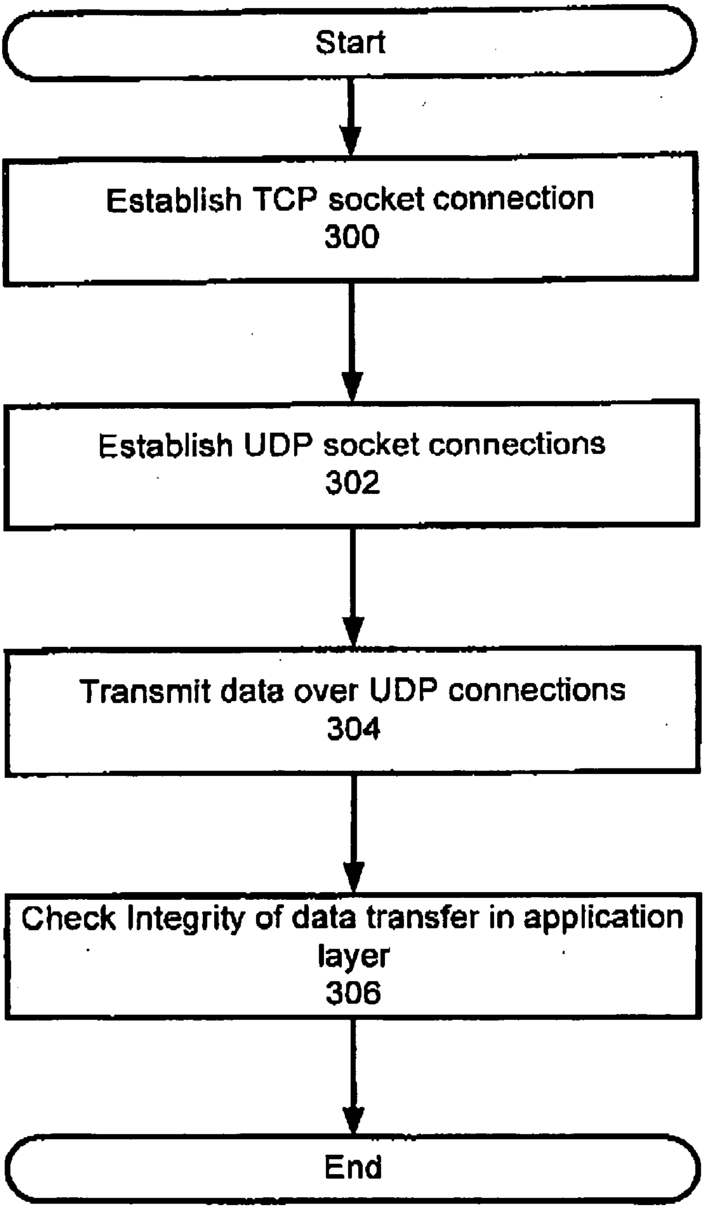

Start
Establish TCP socket connection
300
Establish UDP socket connections
302
Transmit data over UDP connections
304
Check Integrity of data transfer in application layer
306
End

Figure 4

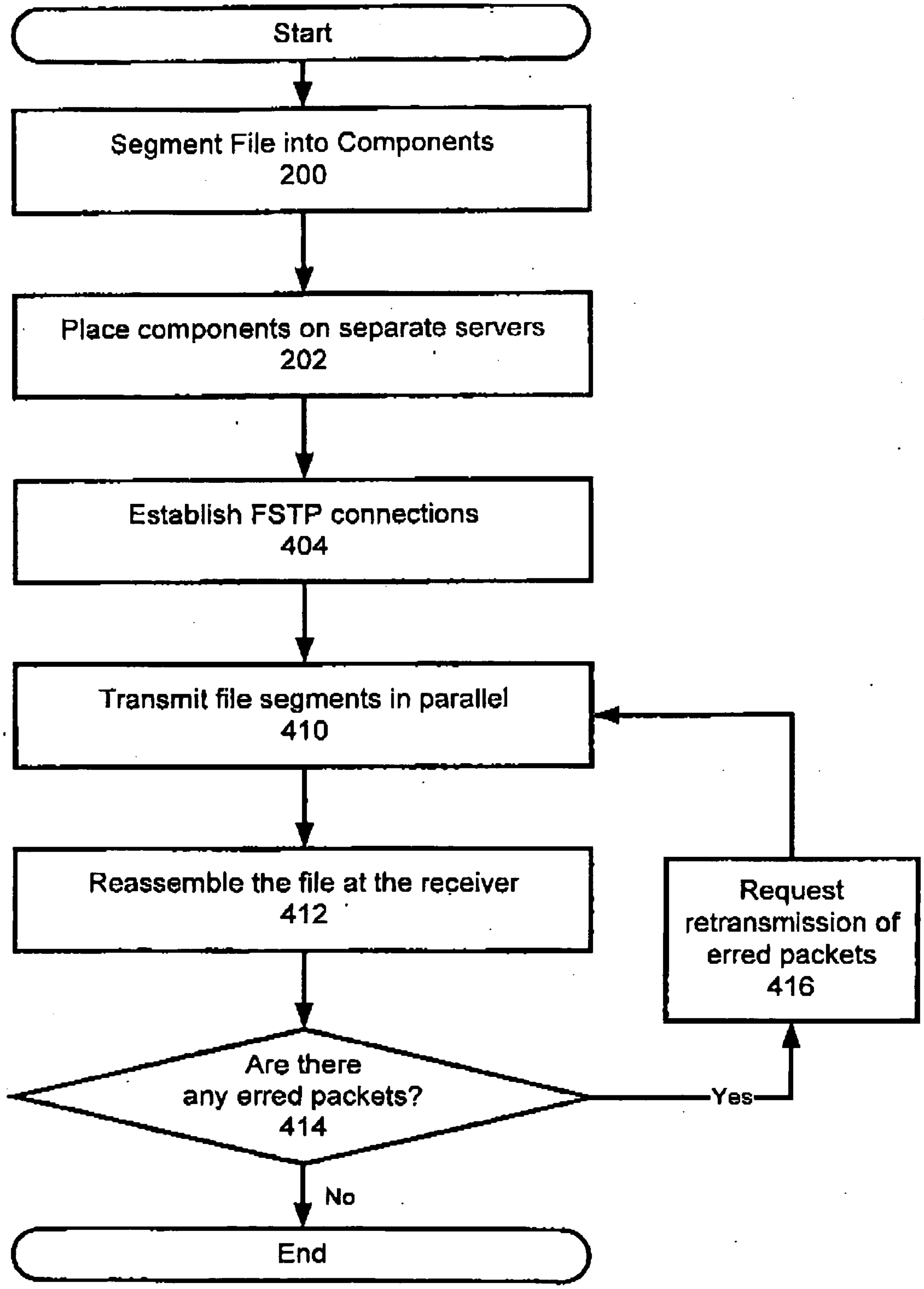

Start
Segment File into Components
200
Place components on separate servers
202
Establish FSTP connections
404
Transmit file segments in parallel
410
Reassemble the file at the receiver
412
Are there any erred packets?
414
Yes
Request retransmission of erred packets
416
No
End

Figure 5

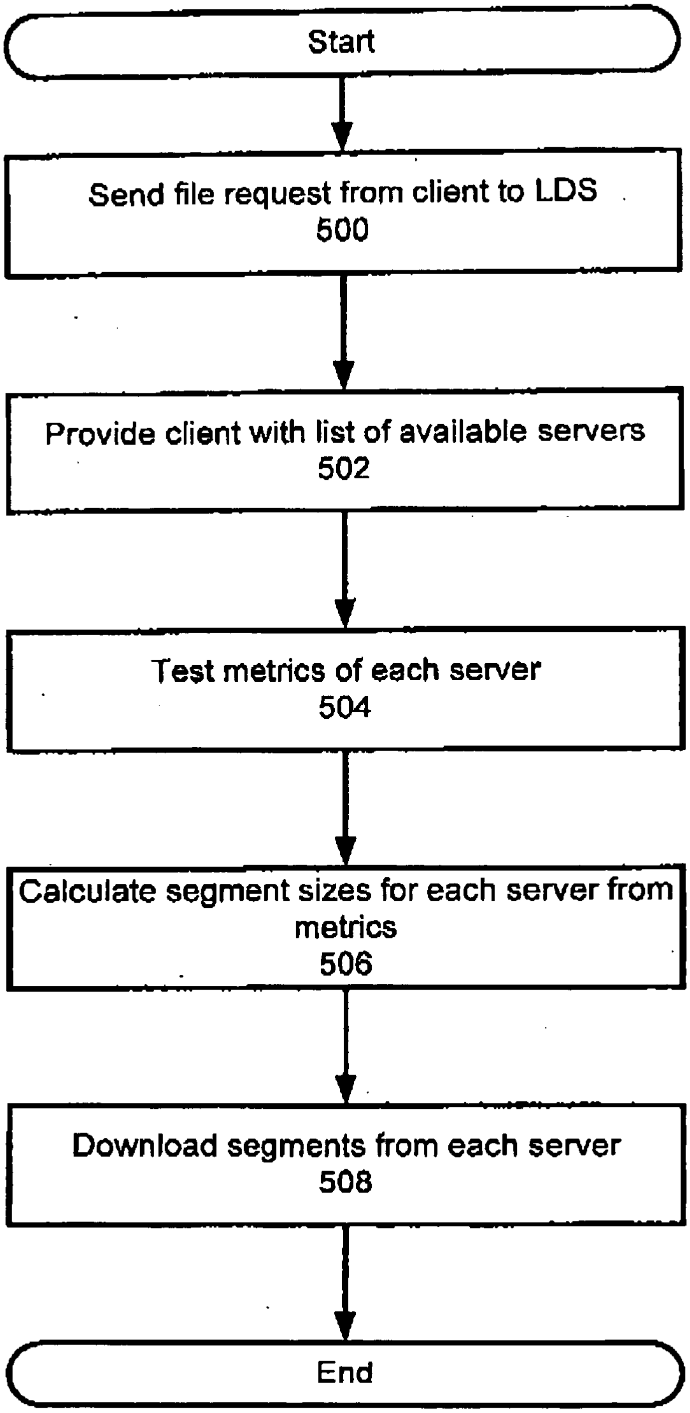

Start
Send file request from client to LDS
500
Provide client with list of available servers
502
Test metrics of each server
504
Calculate segment sizes for each server from metrics
506
Download segments from each server
508
End

Figure 6